PAGE 34/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

EXHIBIT D

The AdviceNet RELEVANCE Language

Universe Communications, Inc.
2180 Dwight Way, Suite C
Berkeley CA 94704

9/18/97

Abstract

The RELEVANCE Language is a key component of the ADVICENET system, enabling authors of advisories to specify with precision the computers for which a certain advisory is relevant. This document describes the formal structure of the language and places in context the many considerations which have led to the current structure.

Confidentiality Notice. The information contained in this documentation is Confidential and Proprietary. It is not for distribution outside of Universe Communications, Inc., which holds the Copyright on the document and all trademarking and patent rights covering names, concepts, and procedures contained inside.

BEST AVAILABLE COPY

22 PAGES

10/20/2004 WED 17:13 FAX 650 474 8401 GLENN PATENT GROUP 034/086

PAGE 35/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

Contents

10/20/2004 WED 17:13 FAX 650 474 8401 GLENN PATENT GROUP 035/086

PAGE 36/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

1 Preliminaries

This document describes the RELEVANCE language of ADVICENET . It aims only to provide basic information about the structure and capabilities of the language. It represents version α .10 and was authored on 9/18/97 .

A full appreciation of the language's use requires an understanding of the ADVICENET system, and can be obtained by careful study of the following documents.

- THE ADVICENET SYSTEM. Describes the current crisis in computer reliability and maintenance. Describes the ADVICENET system for communicating active advisories to computers worldwide. Describes the impact of this system on components of the existing crisis. Describes the development effort needed for a computer software/hardware manufacturer or a corporate intranet adminsitrator to benefit from ADVICENET .
- THE ADVICENET SITE DEVELOPER'S MANUAL. Describes the components of an advice site and how an advice provider can construct those components. Illustrates how an advice site can save an advice provider money in technical support and maintenance costs.
- THE ADVICENET INSPECTOR API. Describes how an advice provider can extend the RELEVANCE language to include capabilities directly addressing specific needs of the advice provider. Gives detailed information on the Applications Programming Interface and on the programming environment which is required to develop Inspectors.
- THE ADVICENET MAC OS INSPECTOR LIBRARY. Describes a layer of Macintosh-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a MacOS computer.
- THE ADVICENET WIN95 INSPECTOR LIBRARY. Describes a layer of Windows '95-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a Wintel '95 computer.

These documents are currently still under development.

2 Role of the RELEVANCE Language in AdviceNet

To set the stage for a detailed description of the RELEVANCE language, we begin with a sketch of the role played by the language in ADVICENET .

The ADVICENET system is based on the idea of a decentralized community of advice providers offering *advice sites* distributed worldwide throughout the Internet. These advice sites make available for HTTP access special documents called *advisories*. Advisories are ASCII text files resembling very much in format existing e-mail messages; in fact they are custom extensions of the e-mail/MIME message formats offered through Internet Standard RFC 822. It is expected that at some future date, the ADVICENET extensions will have proven so widely useful that they will become Internet standards themselves.

Advisories have a format resembling the format of e-mail messages, with many of the same components in the message/digest headers. The key extension offered by advisories

10/20/2004 WED 17:13 FAX 650 474 8401 GLENN PATENT GROUP 036/086

PAGE 37/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

is the institution of a new clause in the message: the *relevance clause*. The relevance clause is distinguished by the keyword `X-Relevant-When:`; what follows the keyword is an *expression* from the RELEVANCE language. It is this language that is being described in this technical report.

The purpose of this language is to enable the ADVICENET *Advice Reader*, running on a user's computer to *automatically* read an advisory and determine, *without intervention from the user*, whether the advisory is *relevant* to the user. Under the mediation of various user interface parameters set in the Advice Reader, the user will typically be notified of relevant advisories, and will be given the opportunity to *follow-through* as the advisories suggest. The actual follow-through action will depend on the advisory, but typically this will be an option to obtain and execute an application which renders, in a fully-automatic way, a repair to an undesirable situation.

This scheme means that it is possible for an advice author to publish advisories at its web site that solve many of the most common technical support problems faced by clients of the author's organization, and be assured that the advisories come to the attention only of users of precisely the computers that are in need of the specific advice being offered. The value in this procedure is that it saves humans the time and trouble of precisely matching solutions to computers in need of those solutions.

With this background, the RELEVANCE language can be seen as fitting into a context of

(i) Widespread distribution via Internet;

(ii) Unattended automatic parsing and evaluation;

(iii) User notification of *relevant* advisories only.

It is particularly important to keep in mind that advice is evaluated for relevance *continually*. That is to say, andvisory, once loaded into a computer's advice reader, can reside in the advice reader for an unlimited amount of time, and can continue to be checked day after day, imposing a de facto constant vigil watching the system state for conditions as they develop. Hence it is particularly important that the system be designed with unattended operation in mind, and that unatteded operation be reliable and even confidence-inspiring.

3 Design Goals of AdviceNet

The RELEVANCE language has been very specifically designed with this application background and purpose in mind. The design as it stands today aims at combining the following characteristics.

- *English-Accessible.* A relevance clause is, in principle, something an individual user could read and comprehend, though few users will choose to do so in most cases. The syntax of the RELEVANCE language resembles the syntax of plain english, with key roles in the language played by clauses formed from articles like *of*, *as*, *whose*, *exists*, and so on.

- *Descriptive.* The purpose of a relevance clause is to examine the state of an individual computer, and see whether it meets various conditions which could combine to imply the relevance of a certain advisory. In short the purpose of the language is to describe rather than to manipulate.

10/20/2004 WED 17:14 FAX 650 474 8401 GLENN PATENT GROUP 037/086

- *Nonprocedural.* Unlike many languages used in connection with the operation and/or maintenance of computers, the RELEVANCE language is not procedural: it does not specify how to manipulate the contents of various fragments of memory. This is the flip side of being descriptive: there is no useful purpose that would be served by enabling traditional procedural services: loops, assignments, and conditionals. Such services are not made available.

- *Open-ended.* The purpose of the language is to precisely describe the state of a computer. This state can change as the user purchases new software and/or hardware, or indeed as new software/hardware objects are invented. Consequently it is not possible to limit in advance what are all the components of state that the language will give access to; and the language is designed to give authors the ability to extend the language to express concepts about system state that haven't yet been conceived of.

- *Object-Oriented.* The language is extensible using the discipline of object oriented programming (OOP). The OOP buzzword means, in our case, that we offer a *Strongly-Typed, Polymorphic* language. We follow the strong-typing philosophy in the extreme. We carefully define the collection of all objects one would like to query, and carefully specify the methods for making queries, and forbid combinations that cannot be formed using our controlled collection of methods. We aim to avoid the usual procedural approach offering powerful, flexible, general purpose tools which can be used in many unanticipated ways. In the usual cases where that approach is used, it is good, but in the setting we are studying, it would be *dangerous*. OOP methodology allows us to design a controlled environment enabling powerful queries of the system state while remaining relatively secure from misuse and abuse.

- *Security and Privacy Aware.* Because the language is designed to be used in conditions of unattended operation over the Internet, it is very important that the language be safe in various ways. This concern has overwhelmed all others in the design of the product; it is fair to say that each of the distinctive features of the RELEVANCE language has been chosen based on these important concerns. It is believed that the language itself best addresses the security and privacy concerns that Internet operation will demand.

4 Security Issues in AdviceNet

The Internet is not a secure medium; along with the many services which it provides come the risks of exposing one's machines to potentially erroneous or even malicious agents. In designing the ADVICENET system, with its aspects of decentralized authorship and unattended operation, it is crucially important for the acceptance and usefulness of the system to address Internet security concerns.

We begin with the obvious comment that there is no such thing as an "ironclad" guarantee of security. The RELEVANCE language has been designed with several security issues in mind; the design addresses these issues, though there remains the possibility of other concerns we have not thought of. Your mileage may vary.

Our main concern is that the evaluation of clauses in the language not have damaging or embarassing side-effects. In evaluating clauses of the language, ideally we would have these four properties:

PAGE 39/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

1. *No significant Claim on system resources.* Evaluation of a relevance clause should not consume inordinate amounts of CPU time, CPU memory, or hard drive space.

2. *No permanent change in system state.* The evaluation of clauses cannot effect the system, for example through creating or destroying files.

3. *No violation of user privacy.* The evaluation of clauses cannot result in an advisory communicating private data about the user to the outside world.

4. *Robustness against misuse of the language.* Desiderata 1-3 are not only supposed to hold in evaluation of well-formed clauses authored by those with "friendly intent". The evaluation of *arbitrary* clauses, even those which are malformed or which are authored by agents with "hostile intent", should *never* be able to lead to damaging side-effects. For example, the evaluation should be safe despite subtle mis-uses of the language – improperly formed clauses, misapplied operators, or mismatching data types.

These different concerns have dictated various aspects of the language design. For example, the RELEVANCE language is nonprocedural; this responds to security concerns, as it places limits on the resources which evaluation can consume. At an elementary level, the RELEVANCE language does not allow for recursion, for infinite loops, for arbitrarily-sized arrays to be allocated; these limits allow explicit bounds to be known (in principle) about the resources which a relevance clause can consume. A famous problem in logic concerns the ability to predict whether a machine evaluating an expression in a procedural language will *ever* reach completion. This is the *Turing Halting Problem*. It is not in general possible to decide whether a program in a procedural language will ever terminate. The situation in our setting stands in stark contrast:

All RELEVANCE *expressions are decidable: they must halt.*

The object-oriented nature of the language also responds to security concerns. As the language is strongly-typed, it is not possible to apply operators to data which they were not designed to support. This allows to avoid crashes and dangerous misreferences, and to safely terminate the evaluation of expressions that are poorly formed.

A second facet of the object-oriented design which may be viewed as a security feature is the fact that manipulations of data are obtained not in the *surface level* of the RELEVANCE language itself but rather in the deeper level of the member functions of data types. These can be expected to be programmed more carefully than would be customary in the programming of a script, and to pay closer attention to clever use of computational resources. Moreover, they provide services which are available only after they are explicitly installed by the user, who therefore has a chance to apply scrutiny to the reliability and authenticity of the provider.

In general, our goal is that the user of ADVICENET should face no more exposure to security risks than a user normally suffers through routine Internet activity. We believe that in many ways ADVICENET creates fewer risks than existing widely used technology such as Web Browsers, in the forms in which such technology is often used (e.g. with "Cookie Dropping" enabled).

10/20/2004 WED 17:14 FAX 650 474 8401 GLENN PATENT GROUP 039/086

PAGE 40/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-60

5 Privacy Issues in AdviceNet

Modern PC's contain an enormous amount of information about their users, some of this highly sensitive. It is very important for the acceptance and usefulness of ADVICENET that we aim at the goal

Information on the machine stays on the machine.

In accessing advice and subsequent evaluation of relevance clauses we aim at two key principles

1. *Subscriber Anonymity.* No one should have a systematic way to know who is subscribing to a given advice site; in the standard ADVICENET system, it is not consider acceptable behavior to require the names and addresses of subscribers.

2. *No Feedback.* No one should have a systematic way to know which advice is relevant on which machines. There is nothing in the evaluation mechanism which can provide information about the user's machine to any other party.

Point 2 is particularly worth examining. The lack of feedback remains true *even if a malicious party has published libraries which communicate using the Internet.* Because the RELEVANCE language has a strict object oriented definition, and all object references lead to function calls with literal constants, it is not possible for a relevance clause to evaluate a function with an argument that is the result of an expression. Therefore even security holes cannot lead to publication of private information.

We aim for no more exposure to privacy risk than a user normally suffers through routine Internet activity.

6 More On Security and Privacy

The main ingredient of good security is the user's common sense: he/she should deal only with trusted, responsible advice providers. As described in the document *The AdviceNet System*, there are various ingredients in the system which, supported by trust mechanisms, deliver reasonable expectations of security.

An important ingredient is the division of labor between RELEVANCE language and the rest of the system. The RELEVANCE language does not offer the ability to fix problems, only to identify them. Fixing problems can only be done upon user approval. This gives a layer of insulation and a chance for reflection. The user is free to not approve that certain tasks be done; the user is free to study the situation before heeding an advisory, and in particular to inquire whether the advice itself is still being offered by the trusted site that first provided it, and whether the advice really was provided by that site. By these mechanisms, it is possible to protect trusted sources of advice, avoiding contamination by outside advice, pretending to be from a trusted source.

A further very important component is the recursive nature of the advice system itself. It is possible for an author to publish advice disowning earlier advice from the same author, recommending that it be removed from one's system; it is also possible to publish advice recommending not to heed advice from conflicting or dubious sources.

In short, the ADVICENET system contains a variety of tools addressing security issues; it is not the responsibility of the RELEVANCE language alone.

10/20/2004 WED 17:15 FAX 650 474 8401 GLENN PATENT GROUP ☑040/086

PAGE 41/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

It must also be admitted that there *are* security and privacy risks in ADVICENET. However, these essentially do not involve the RELEVANCE language.

The greatest *security* risks involve what happens when a user decides to follow through on a piece of advice. At that point, in principle, anything is possible; when the user says "OK" he/she is effectively opening the machine to the advice provider who is then in a position to perform surgery of whatever magnitude.

It is likewise true that user approval poses the greatest *privacy* risks. The application that runs in response to user approval has, in principle, the ability to communicate to the Advice Provider, and there are no real restrictions on what can be communicated. An Advice Provider *can* know if a certain action (update or modification) something was approved by a certain user at a certain time. This can reveal useful information to the provider, since the provider is learning that a complicated condition on the system state is true. Depending on the complexity of the condition, the information revealed may be substantial.

Although the security and privacy risks that would be posed by improper design of the RELEVANCE language are not serious compared to the risks posed by indiscriminate user approval of untrusted follow-through, it is important to pay careful attention to these risks. It can be anticipated that ADVICENET will parse and evaluate many tens of thousands of relevance clauses for every clause that is found to be relevant. Hence many orders of magnitude more work will be done without user intervention than with user intervention. Risks posed by the evaluation process, even if suffered only rarely, might potentially have a numerically far more important role than other risks posed by the system.

It is also important to point out that in special circumstances, one might *not* want the security and privacy features to be available. One can anticipate that in a secure intranet setting, barred from outside access by a firewall, it would be useful for intranet managers to be able to know which machines on a corporate network respond 'Relevant' to certain queries. In that event, it would be useful to disable some of the restrictions we have placed on the language in the design discussed here. Our design goal emphasizes security and privacy, but in certain versions of the final product other emphases will be possible.

One can also imagine that certain advice providers will want to know who is subscribing to advice because they are selling advice and need to make it available only to paying customers. This arrangement violates the stated privacy goals underlying this document. However, while we emphasize privacy goals, we will also enable applications in which such goals are secondary.

7 Properties of the RELEVANCE Language

We now describe important details about the language structure and function. Given the earlier discussion, certain of the information presented here will appear repetitive; it seems best to be absolutely explicit about many details at this stage, and this seems to require repetition.

7.1 Descriptive Language

The RELEVANCE language is used for querying the state of a computer, a highly complex arrangement of software, hardware, and data. It is a description language which describes state rather than a procedural language which describes actions. As the language is not used

10/20/2004 WED 17:15 FAX 650 474 8401 GLENN PATENT GROUP 041/086

PAGE 42/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

for traditional procedural tasks (e.g. sorting data, moving data) it is *profoundly* different from a traditional procedural language. There are

- no named variables
- no assignment statements
- no function calls, or at least no explicit function calls with variable arguments
- no loops or conditional execution

Nevertheless, the language is designed to be powerful, in that it is intended to be *highly expressive*; a few words in this language provide access to answers about the system state which would be impossible to obtain in traditional programming languages short of writing hundreds of lines of code and invoking many specialized functions in system libraries.

Certain tasks can be done either in a descriptive language or in a procedural language; the code in a RELEVANCE language will typically look very different than the code for the same task in a procedural language, and the way of productively thinking about how to accomplish the same task will be very different between the two languages. It will be important for some programmers to remind themselves from time to time the main reasons why a relevance clause *should* be very different from a program in a procedural language. Three good reasons are

- Because procedural langauges pose security & privacy concerns;
- Because descriptive languages can be more economical (in terms of code length) for the purposes of writing relevance clauses;
- To make the distinction in purpose starkly obvious to programmers.

7.2 Layered Language Definition

It must be understood that the RELEVANCE language is very open-ended, and that it is built up in layer upon layer of extensions. In this document we only give detailed information about the base 'built-in' layer; with a few hints about other layers.

It must also be understood that the layers interface via object-oriented programming techniques. Hence the capabilities of each additional layer are delivered in packages "plugging-in" new object-oriented deliverables. In the RELEVANCE language these are data types, data properties, data elements, data casts, and operators.

Hence, to understand a completely installed system is to understand the layers which have been installed, and to understand the object-oriented services that each layer provides.

We envision that in a typical installation, these layers will go as follows:

- *Base Layer.* Contains the basic mechanics of clause evaluation: the Lexer, and the Parser for the language syntax, along with a number of Basic Built-in types, properties, elements, casts, and operators. It is expected that the base layer will be the same on every platform carrying ADVICENET .
- *System-Specific Layer.* This consists of Plug-ins which give basic system information about a certain family of computers. For example, one might be able to get the system date and time, the size of various files, the contents of the PRAM, or the names of attached peripheral devices. It is expected that these will all be in common

10/20/2004 WED 17:16 FAX 650 474 8401 GLENN PATENT GROUP 042/086

PAGE 43/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

for a given computer OS family – Win '95, MacOS – independent of manufacturer. It is expected that these will be similar from one OS to another – the object oriented structures and queries being basically the same – but perhaps with naming difference (e.g. "Folder" on one OS might be "Directory" on another).

- *Vendor-Specific Layers.* This collection of potentially a large number of "extensions Layers" would typically be produced by third parties giving special access to the internals of their hardware and software products: one can think of potential authors ranging a span of products from hardware producers (of, say, Cable Modems) to software producers (of say Photoshop & Plug-Ins); to service providers (e.g. America On-Line).

Define Primitives Contents

It is expected that 80% or more of the power of the RELEVANCE language will be provided through services offered outside the Base layer.

We do not at the present time have any thoroughly thought-through examples of Class Libraries outside the Base layer. Obviously this is an important next step, which we expect to describe in two documents:

- THE ADVICENET MACOS INSPECTOR LIBRARY. Describes a layer of Macintosh-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a MacOS computer.

- THE ADVICENET WIN95 INSPECTOR LIBRARY. Describes a layer of Windows '95-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a Wintel '95 computer.

7.3 Expected Conditions of Use

We anticipate that many advice providers will use the RELEVANCE language under the following conditions.

- A programmer will author advisories containing relevance clauses.
- Relevance clauses will typically use types, operators, elements, properties from existing inspector libraries to obtain their expressive power.
- Very occasionally, the programmer will discover that a key property (say of a hardware product) is not made available through existing libraries, and will author such a property plug-in.
- The programmer's advice will depend on the existence of various plug-ins for its well functioning – both widely available pre-existing class libraries and his own newly-developed plugins. He will have the responsibility to verify that libraries are correctly installed. It is necessary to write meta-advice to verify this.

Thus the programmer has these responsibilities:

- To know the base language and system-specific inspector library;
- To write the relevance clauses for his advice using those inspectors;

10/20/2004 WED 17:16 FAX 650 474 8401 GLENN PATENT GROUP ☑043/086

PAGE 44/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

- (Occasionally) to extend the collection of existing types and properties;
- To write meta-advice verifying that the language is properly configured to correctly evaluate his advice.

8 The Object Model

While a computer language involves both syntax and a collection of semantics for manipulating data structures, we have chosen in this document to begin with the role of data structures and semantics, and later to discuss syntax.

As discussed earlier, the RELEVANCE language is object-oriented, which means in our case that

- The language is *strongly-typed*: every object has a specific type, and operators and properties may only be applied to objects of appropriate types. (In some non-object-oriented languages, it is possible to misuse functions, applying them to data for which they cannot work properly. This is not possible in the RELEVANCE language).
- The language is *polymorphic*: the meaning of an operator or property reference depends essetially on the type of the direct object. This is also called *overloading* of operators & properties: the interpretation of an operator or property depends heavily on the data to which it will be applied.

The language differs from some other OOP languages in that the concept of encapsulation is de-emphasized. There is no distinction between private, public, and protected data. Or, more precisely, all data are public.

In the current revision of this document, Version α .10 , the important object-oriented notion of *inheritance* will not be discussed; although it is expected to ultimately be included in the system. See Section 8.4 below.

8.1 Objects and Methods

An *object* in the RELEVANCE language may be thought of as a reference to a physical object, such as a *file* or *SCSI device*. It is implicitly a data structure, to which can be applied access methods made available in the RELEVANCE language.

Every object has a *type*, and each type has associated with it several *accessor methods*. The language has four base types, which are essential to the workings of the language interpreter, and many extension types which are essential to the performance of useful work.

The base types are

- *Integer*. A signed 32-bit integer.
- *String*. A variable length character string.
- *Boolean*. A single bit.
- *World*. An abstract type, representing the computer of which the language is running.

The RELEVANCE language accessor methods can be thought of as giving answers to structured queries about objects. They are strongly-typed: an accessor may only be applied to a type for which a definition of that accessor for that type has previously been registered.

Accessors themselves have various classifications:

10/20/2004 WED 17:17 FAX 650 474 8401 GLENN PATENT GROUP 044/086

PAGE 45/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

- *Properties*: methods which inspect an object and return a value.
- *Elements*: methods which inspect an object and under the control of a single parameter, return a value. Often these are components of the object's structure and hence are called elements. *Element-by-iIndex* takes an integer; *Element-by-Name* takes a string.
- *Operators*: methods which inspect an object (unary operators) or a pair of objects (binary operators) and return a value. These are invoked syntactically by the usual arithmetic and relational operators as commonly encountered in high-level languages; but there is no requirement that actual semantics be familiar. The interpretation of `a + b` need not resemble the usual interpretation – if a and b are not of integer type.
- *Casts*: methods which inspect an object and return a new object with a different type or, in some cases, the same type but a different format.

The purpose of the type-method system is to make available powerful query and manipulation tools but only under strong-typing restrictions that inhibit misuse.

8.2 Under The Table

The RELEVANCE language object system is intrinsically connected with the object system provided by C++.

Each Type, Operator, Property, Element, and Cast defined in the language corresponds to a class, an operator, or a public member function in C++

In fact, extensions to the base set of types are provided by writing appropriate C++ code; see the document THE ADVICENET INSPECTOR API for an explanation how this is done.

We can view the library of extension types as providing a rich library of C++ routines for querying the system state and for manipulating the answers that get returned.

We can therefore view the RELEVANCE Language as a tool for providing access to such a library while imposing strong-typing restrictions which inhibit abuses.

8.3 The Object Universe

The bulk of the functionality of the RELEVANCE language is provided by the extension types which have already been mentioned. These can be conveniently organized as a mathematical graph structure, in which *nodes* represent *types* and *arrows* represent *accessor methods* which return properties, elements, or casts of those types.

The structure is rooted at `World`, which has many properties. On the Macintosh, a few of these are:

Properties of World

- System Folder. Returns an object of type *Folder* associated with the system folder.
- Boot Drive. Returns an object of type *Volume* associated with the boot volume.
- Processor. Returns an object of type *CPU* associated with the computer.
- Printer. Returns an object of type *Printer* associated with the currently chosen printer.

10/20/2004 WED 17:17 FAX 650 474 8401 GLENN PATENT GROUP 045/086

PAGE 46/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

- Monitor. Returns an object of type *Monitor*.
- IPAddress. Returns an object of type *IPAddress*.

We can list a few elements as follows:

Elements of World

- SCSI Bus *Integer*. Returns an object of type *SCSI Device* associated with the specified SCSI Bus slot.
- Serial Port *Integer*. Returns an object of type *Serial device* associated with the specified SCSI slot.

It should be clear that, in general, there is a close connection between methods and the types that they return. Often these even have the same name. Remember, though, that the method that has name 'Processor' obtains a property of *World*, not of *Processor*.

The methods applicable to *World* generate a series of types with their own methods; those methods in turn generate other types. For example, an object of type *Volume* has properties *Driver* and *File-Allocation-Blocksize* and *Capacity* and *FreeSpace*, while an object of type *CPU* has properties *Name*, *Speed*, *FPU*, *MMX*, and so forth.

Some of these objects are of atomic type: *Speed* when applied to an object of type *CPU* yields an object of integer type; while *Volume Name*, when applied to an object of type *Volume* yields an object of type string. However, some of these objects are of new type: witness *Driver*, which is an object with properties such as *Vendor* and *Version*.

In a future version of this document we expect to work out in detail all the types and methods associated with two areas of special interest:

1. Internet Settings example
2. Volume examples

8.4 The Evolving Universe

The object model we are describing here is still under development. Two major enhancements to the system described here are currently under development.

- *Inheritance*. A basic platform-independent type (but with platform-dependent implementation) is *File*. This is an object with the following properties on MacOS:
  - Name. Object of type *Filename*.
  - Creation Time. Object of type *Time*
  - Modification Time. Object of type *Time*
  - Length. Object of type *Integer*.
  - Checksum. Object of type *Integer*.
  - Parent Folder. Object of type *Folder*.
  - Version. Object of type *Version*.
  - Type. Object of type *MacFileType*.
  - Creator. Object of type *MacFileCreator*.

10/20/2004 WED 17:17 FAX 650 474 8401 GLENN PATENT GROUP 046/086

PAGE 47/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

Under MacOS, there are also files with special properties, and hence having their own specialized types: *Application*, *Control Panel*, *Extension*. For example, an application has all the properties of a file as well as a *Partition Size*, *Recommended Partition Size*, *Minimum Partition Size*, and *Bundle Bits*. A control panel has all the properties of file, as well as the property *Enabled* (a Boolean).

In traditional OOP systems, one would handle such issues by creating a base type *File* and derived types *Application*, *Control Panel*, etc. The derived types accept all the same accessor methods as the base type *File*, and objects of type *Application*, say, accept also the accessor methods for that type, such as *Partition Size*. We expect that the RELEVANCE language will ultimately do this, though the details remain to be worked out.

- *Plural Properties*. It is important for certain purposes to know that there only be one of a certain object in existence, while for other purposes, it does not matter if there be one or more than one instance matching a reference.

  As a convenient way to make it possible to verify either circumstance, the RELEVANCE language will offer plural properties. For example, the property *application* is singular and the property *applications* is plural. The distinction is this

  - `application "Netscape"` asserts the existence of *exactly one* application named Netscape in the scope. Clause evalaution will generally fail if there is no such application or if there are more than such application (see Section 9.2) for information about the mechanism behind this.
  - `applications "Netscape"` asserts the existence of *one or more than one* application named Netscape in the scope. Clause evaluation will generally fail only if there is no such application (see Section 9.2).

  The existence of singular and plural properties means that phrases have (by inference) singular or plural characteristics. Accomodating this will require certain modifications to the grammar and semantics of the language, which remain to be worked out.

8.5 Describing the Universe

In order to keep track of all the types and methods available in ADVICENET , we will make available a *Type Inspector*, a specialized dictionary indicating all the available types and their associated methods and the results of thse methods.

This dictionary will be rendered in two different ways. In an on-line version, it will offer a hypertext-format dictionary describing each relevant term and the associated definition and relationships. This will be reminiscent, for some readers, of the Class Dictionary in AppleScript [1].

In a printed version, it will take a standard format reminiscent from typical system manuals for UNIX and related operating systems. Roughly speaking, we expect a single-page description of each type, taking the form

10/20/2004 WED 17:18 FAX 650 474 8401 GLENN PATENT GROUP 047/086

PAGE 48/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

Typename – *1-line synopsis*

Description. - 1 paragraph of text describing the general role of this type, and important considerations for those making use of it.

Contained-in. – Name of library that installs it. URL of library source. Current version of that library.

Properties.

Prop1 Value-Type. Description of what the value is supposed to be, and the type that it takes.

...

Elements.

keyword1 name Value-Type. Description.

keyword2 integer Value-Type. Description.

...

Unary Operators.

unop1 Value-Type. description.

...

Casts.

as Typename1 description

...

How-reached. *Description of Types from which this type is reachable as the value of a property or element.*

property1 of Type1

...

Related Types. *Description of Types which have intimate connections with the current one, either*

Type1 Reason.

...

Subtleties. *Hairy points to keep in mind.*

1. Note1.

...

10/20/2004 WED 17:18 FAX 650 474 8401 GLENN PATENT GROUP 048/086

PAGE 49/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

9 The Evaluation Model

We now turn to rules for successful interpretation of clauses in the RELEVANCE language.

9.1 Abort, Retry, Fail

The basic purpose of the RELEVANCE language is to successfully lex, parse, and evaluate a single expression, resulting in a value of type *Boolean*. The resulting value is passed to the ADVICENET advice reader for further processing.

Of course, there will always be cases where a RELEVANCE clause is malformed or otherwise unacceptable. In order to deal with that situation in an organized fashion, the lexer/parser/interpreter follows this rule.

> A RELEVANCE Clause will lead to an advisory being declared relevant *only* if, the clause is *intelligible*, *successfully evaluates* to a result of type *boolean*, and is *true*.

Let's explain this rule in more detail:

- Expressions can be *Unintelligible* or *Intelligible*. *Unintelligible* expressions occur for one of three reasons.
  - *Lexical unintelligibility*. For example, the expression contains bizarre unquoted characters, such as $.
  - *Syntactic unintelligibility*. For example, the expression contains contains disallowed usage, such `++name`.
  - *Run-time unintelligibility*. For example, the expression refers to an unknown property (e.g. `exists solarplexus of file`).
- Intelligible expressions can *Succeed* or *Fail* to evaluate.
  - *Arithmetic Failure*. 65536*65536 overflows type *integer*. → NSO
  - *Reference Failure*. `picture resource 101` doesn't exist.
  - *System Failure*. There is an error reading the hard drive.
- Successful evaluations can be *Boolean* or not. Examples of types which are not Boolean: *integer*, *IPAddress*, etc.

9.2 Exception Handling

The RELEVANCE language interpreter creates two special objects to indicate evaluation failures.

- NSO, which stands for 'No Such Object'.
- TMO, which stands for 'Too Many Objects'.

For example the expression `length of file "foo"` evaluates to NSO if file `"foo"` doesn't exist.

NSO *generally* but not always causes expressions to FAIL to evaluate. There are at the moment two exceptions:

1. `exists(NSO)` evaluates successfully.
2. `exists folder whose ( length of file "foo" of it is 8 )` evaluates successfully if SOME folder contains a file `"foo"`, even though many folders do not.

The expression `application "Netscape"` evaluates to TMO if there are *two or more* applications which are instances of Netscape.

TMO *generally* but not always causes expressions to FAIL to evaluate. There are at the moment two exceptions, analogous to the exceptions for NSO discussed above.

10/20/2004 WED 17:18 FAX 650 474 8401 GLENN PATENT GROUP 049/086

PAGE 50/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

10 RELEVANCE Language Syntax

We finally attempt a description of the syntax of the RELEVANCE language!

10.1 Simple Examples

1. Existence of a certain application.

```
X-Relevant-When: exists application "Photoshop"
```

2. Comparison of version numbers.

```
X-Relevant-When: exists Control Panel "MacTCP" and
                 version of Control Panel "MacTCP" is version "2.02"
```

3. Compare modification dates.

```
X-Relevant-When: exists Photoshop PlugIn "Picture Enhancer" and
                 modification time of Photoshop PlugIn "Picture Enhancer" is
                 greater than time "10 January 1997"
```

4. Examine Control Panel Settings.

```
X-Relevant-When: exists Control Panel "ConfigPPP" and
                 Transport Mechanism of Control Panel "MacTCP" is equal to "SLIP"
```

We now turn to a more systematic description of the language; we consider in turn: Lexical analysis, Syntax, and Semantics.

10.2 Lexemes

The lexical analysis of the string breaks up the input string into tokens consisting of basic elementary inputs.

In the following, we will denote literal keywords by enclosing them in curly braces {`like this`}, and we will denote general Lexeme Classes by [class name].

Lexical tokens come in one of the following basic categories.

[String] A string of printable ascii characters enclosed in quotation marks (").

[Integer] A potentially signed string of decimal digits.

[Minus] The character -.

[SumOp] The characters +-.

[PrdOp] The characters */.

[RelOp] The character sequences = > >= <= !=.

[Phrase] A sequence of one or more unquoted words.

These categories are mostly unexceptional, and will be familiar to programmers from work with other computer languages.

10/20/2004 WED 17:19 FAX 650 474 8401 GLENN PATENT GROUP 050/086

10.2.1 Phrases

The most non-standard aspect of the language is in the definition of token type [Phrase]. Formally, a Phrase is a string of words; a word is a string of characters beginning with a letter. Any Phrase can be further decomposed into several occurrences of [Reserved Phrase], with, intervening word sequences (by definition) called [Ordinary Phrase].

Here is a partial list of
Reserved Phrases.

- as
- and
- containing
- ends with
- exists
- is
- it
- number of
- or
- remainder
- starts with
- whose

These phrases are all built into the base layer of the language, and have purposes associated with the base functions of the language; hence they are reserved from other use.

Here is a partial list of
Ordinary Phrases.

- Control Panel
- Photoshop PlugIn
- Modification Time
- MacFileType
- MacFileCreator
- SCSI Bus
- Shared Disk
- CD ROM

These are all phrases that describe system-dependent concepts, and so they are associated with the System-specific or Vendor-specific layers of the language. The collection of ordinary phrases is not fixed in advance, and can be exepected to grow with time as more inspectors are added to the language.

We note that one has *Run-Time Unintelligibility* when the lexer extracts an ordinary phrase from a string of words and cannot find that phrase in the current collection of installed libraries.

PAGE 52/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

10.2.2 Special Notes

1. Strings are simply quoted sequences of ASCII characters.
   - There is no a-priori length limit on strings. It is a *System Evaluation Failure* if the lexer encounters strings so long that they cannot be stored in available RAM.
   - RFC 822 imposes length restrictions on strings.
   - There will be a mechanism for continuing strigs across lines.
2. Escape sequences will be specified, but have not yet been defined. Most likely usage is to specific ASCII codes explicitly by $\backslash D_1D_2D_3$ where the D_i are decimal digits, although hexadecimal and other coding schemes might be offered. However, issues of accents and other items that appear in connection with foreign character sets will have to be carefully studied.

10.3 Formal Grammar

In the table below, we denote concepts defined in the left-hand-side of Grammar Productions by ⟨name⟩

```
<Goal>          := <Expr>

  <Expr>        := <Expr> {or} <AndClause>  | <AndClause>

  <AndClause>   := <AndClause> {and} <Relation> | <Relation>

  <Relation>    := <SumClause> [RelOp] <SumClause> | <SumClause>

  <SumClause>   := <SumClause> [SumOp] <Product>
                 | <SumClause> [Minus] <Product>
                 | <Product>

  <Product>     := <Product>   [PrdOp] <Unary>
                 | <Unary>

  <Unary>       := [Minus] <Unary>
                 | [UnyOp] <Unary>
                 | <Cast>

  <Cast>        := <Cast> {as} [Phrase]
                 | <Reference>

  <Reference>   := [Phrase] {of} <Reference>
                 | [Phrase] [string]   {of} <Reference>
                 | [Phrase] [integer]  {of} <Reference>
                 | [Phrase] <Restrict> {of} <Reference>
                 | [Phrase] [string]
                 | [Phrase] [integer]
                 | [Phrase] <Restrict>
                 | [Phrase]
                 | {exists} <Reference>
                 | {number of}  <Reference>
                 | [string]
                 | [integer]
                 | {it}
                 | ( <Expr> )

  <Restrict>    := {whose} ( <Expr> )
```

10/20/2004 WED 17:19 FAX 650 474 8401 GLENN PATENT GROUP 052/086

PAGE 53/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

10.4 Special Language Constraints

There are two very important special constraints imposed by the syntax of the language which we now discuss. Each one can be described in terms of what the RELEVANCE language does not allow which programmers may be familiar with due to experience with procedural languages.

10.4.1 Lack of variables in function calls

For the lack of a better terminology, we have chosen to use (archaic) procedural language to describe this special feature of the language. The usage doesn't quite fit because the RELEVANCE language really has no function calls.

Still, we may think of "Element-by-Name" as implicitly invoking a certain function with two arguments the first being an object, the second a parameter that selects the precise object. For example:

Parent of application

```
file "Read Me" of folder "Photoshop"
```

might be coded in a procedural language as

```
for (i=0; i<NVolumes; i++){
   Volname = Volume[i];
   FolderRef = findfolder(Volname,"Photoshop");
   if(FolderRef != 0){
       FileRef   = getfile(FolderRef,"Read Me");
       break;
   }
}
```

Here `folder "Photoshop"` asks for an element-by-name of the World, which is similar to calling, for each mounted volume, the procedural language function `findfolder(volname,name)` with appropriate parameters to have it search a certain volume for a certain folder.

The key point is that the syntax of the RELEVANCE *language forces you to ask for specific elements-by-name using string* constants *for the selectors. It is impossible to perform the analog of calling such a function with a computed name.*

Thus, one can not say

```
file (filename of file 3 of volume "Shared Disk") of
folder "Photoshop" of volume "Private Disk"
```

This is a potentially useful script: one looks on one of volume for a file and looks for a file by the same name on another volume. *It is forbidden syntactically to do such things in the* RELEVANCE *language.*

The rationale for this taboo is the existence of security problems which would inevitably appear if it were tolerated. We expect that for many advice authors there will be a period of psychological adjustment as they adapt to this restriction.

10.4.2 Lack of Nested Loops

Since there are no variables or assignments in the RELEVANCE language, one may wonder at first how one could obtain the equivalent effect of iterating across a list of folders or files

10/20/2004 WED 17:20 FAX 650 474 8401 GLENN PATENT GROUP 053/086

to check a condition. In the RELEVANCE language *loops a single level deep* can be handled by the use of the whose primitive, and the associated `it` reference.

For example, suppose we wanted to find out if the user's Eudora folder has any mail digest files which were last modified before the ship date of the current release of Eudora.

```
exists file of folder "Eudora"
  whose ( modification time of it is less than time "12 January 1997" )
```

Here the whose clause is causing the implicit iteration over all files in the Eudora folder. It is useful to think of whose as creating a function with the single formal parameter `it` and then calling the function once for each file in the indicated folder.

On the other hand, suppose we wanted to find out if the user's Eudora folder has any mail digest files which are not accompanied by table of contents files. We might think of writing

```
exists file of folder "Eudora" whose ( not exists (file (prefix of it & ".cnt")))
```

This pseudo-fragment will not work because it specifies the forbidden usage "file (computed-result)" (see previous subsection)

We might also think of writing

```
exists file of folder "Eudora"
     whose ( suffix of it-1 is ".mbx" and
        not exists file of Folder "Eudora"
            whose (prefix of it is (suffix of it-1 suffix of it-2 is ".cnt") )
```

This psuedo-fragment is an attempt to make nested loops; we would like to have `it-1` be the 'local variable' in the 'scope' of the 'outer whose' and `it-2` be the 'local variable' of the 'scope' associated with the 'inner whose' . Note however that these are vain impulses: they are not allowed syntactically.

11 To Probe Further

- Relation to AppleScript.
- Readings on OOP.

12 Appendix 1. Properties of World

13 Appendix 2. The Base Library of Types and Accessors

References

[1] The Tao of Apple Script.

[2] The Tao of Objects.

[3] THE ADVICENET SYSTEM.

[4] THE ADVICENET SITE DEVELOPER'S MANUAL.

System changes since Qamr TIME!

PAGE 55/86 * RCVD AT 10/20/2004 8:03:33 PM [Eastern Daylight Time] * SVR:USPTO-EFXRF-1/1 * DNIS:8729306 * CSID:650 474 8401 * DURATION (mm-ss):28-50

[5] The AdviceNet Inspector API.

[6] The AdviceNet Mac OS Inspector Library.

[7] The AdviceNet Win95 Inspector Library.

LAST

10/20/2004 WED 17:20 FAX 650 474 8401 GLENN PATENT GROUP 055/086

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ❑ BLACK BORDERS
- ❑ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ❑ FADED TEXT OR DRAWING
- ❑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ❑ SKEWED/SLANTED IMAGES
- ❑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ❑ GRAY SCALE DOCUMENTS
- ❑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ❑ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ❑ OTHER: ______________________

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.